**1.- Dado el siguiente fragmento de código**

if (a and b) {

x=x+1;

}

if (c or d) {

x=x+1;

}

else {

x=x+3;

}

**Teniendo en cuenta que 'A', 'B', 'C' y 'D' representan condiciones booleanas simples y los operadores lógicos and y or se evalúan sin cortocircuito indica como el menor número posible de casos de prueba el valor booleano que deben tener cada una de las condiciones simples en cada caso de prueba para obtener.**

**A) Una cobertura de condiciones del 100% y decisiones del 100% (nvl 4) (Puedes añadir tantas filas como consideres necesarias).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **id** | **A** | **B** | **C** | **D** |
| **1** | **T** | **T** | **T** | **T** |
| **2** | **F** | **F** | **F** | **F** |

**B) Una cobertura de condiciones múltiples del 100% (nvl 5) (puedes añadir tantas filas como consideres necesarias).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **id** | **A** | **B** | **C** | **D** |
| **1** | **T** | **T** | **T** | **T** |
| **2** | **T** | **F** | **T** | **F** |
| **3** | **F** | **T** | **F** | **T** |
| **4** | **F** | **F** | **F** | **F** |

**C) Una cobertura de condiciones del 100% y decisiones del 100% (nvl 4) (Puedes añadir tantas filas como consideres necesarias ESTA VEZ IMAGINA QUE ES CON CORTOCIRCUITO).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **id** | **A** | **B** | **C** | **D** |
| **1** | **T** | **T** | **T** | **T** |
| **2** | **F** | **F** | **F** | **F** |
| **3** | **F** | **F** | **F** | **T** |

**D) Una cobertura de condiciones múltiples del 100% (nvl 5) (puedes añadir tantas filas como consideres necesarias, ESTA VEZ IMAGINA QUE ES CON CORTOCIRCUITO).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **id** | **A** | **B** | **C** | **D** |
| **1** | **T** | **T** | **T** | **T** |
| **2** | **T** | **F** | **F** | **T** |
| **3** | **F** | **T** | **F** | **F** |

**2.- A partir del proyecto multimódulo matriculación visto en prácticas se pide**

**a) Completa la siguiente configuración del pom.xml (rellenando los huecos correspondientes)**

<artifactid>matriculacion</artifactid>

<version>1.0-SNAPSHOT</version>

<modules>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| **<module>matriculacion-comun</module>** |

| **<module>matriculacion-dao</module>**  |

| **<module>matriculacion-bo</module>**  |

| **<module>matriculacion-proxy</module>** |

---------------------------------------

</modules>

<packaging>**pom**</packaging>

<name>matriculacion</name>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactid>**maven-failsafe-plugin**</artifactid>

<version>2.22.2</version>

<executions>

<execution>

<goals>

<goal>integration-test</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

**b) Completa los siguientes test de integración para los métodos AlumnoDAO.addAlumno() y AlumnoDAO.delAlumno(), teniendo en cuenta lo siguiente**

**Disponemos de los siguientes ficheros .xml:**

|  |
| --- |
| tabla0.xml (base de datos inicial en cada test) |
| <?xml version="1.0" encoding="UTF-8">  <dataset>  <alumnos nifs="11111111A" nombre="Alfonso Ruiz" direccion="Rambla, 22" emails="alfonso@ua.es" fechaNac="1980-02-22" />  <alumnos nifs="22222222B" nombre="Laura pErez" direccion="Maisonave, 5" emails="laura@ua.es" fechaNac="1980-02-22" />  </dataset> |

|  |
| --- |
| tabla1.xml (base de datos esperada como salida en testDelete) |
| <?xml version="1.0" encoding="UTF-8">  <dataset>  <alumnos nifs="22222222B" nombre="Laura pErez" direccion="Maisonave, 5" emails="laura@ua.es" fechaNac="1980-02-22" />  </dataset> |

**Los casos de prueba cuya implementación has de completar son los siguientes:**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **MÉTODO A PROBAR** | **ENTRADA** | **SALIDA ESPERADA** |
| testAdd | void addAlumno(AlumnoTo alumno) | alumno=null | DAOException |
| testDelete | void del Alumno(String nif) | nif="11111111A" | Tabla1 |

**Y la implementación a completar (rellenando los huecos correspondientes) se muestra a continuación teniendo en cuenta que los huecos acabados en ';' deben rellenarse con una única sentencia).**

public class AlumnoDAOIT {

private IAlumnoDAO alumnoDAO;

public static final String TABLA\_ALUMNOS = "alumnos";

private IDatasetTester databaseTester;

@BeforeEach

public void setUp() throws Exception {

databaseTester = new JdbcDatabaseTester("com.mysql.jdbc.Driver","jdbc:mysql://localhost::3306/**MATRICULACION**?useSSL=false","root","ppss");

IDataSet dataSet = new FlatXmlDataFileLoader().load(**"/tabla0.xml"**);

**databaseTester.setDataSet(dataSet)**;

**databaseTester.onSetup();**

alumnoDAO = new FactoriaDAO().getAlumnoDAO();

}

@Test

public void testAdd() throws Exception {

**Assertions.assertThrows(DAOException.class,**

**()->alumnoDAO.addAlumno(null)**

**)**

}

@Test

public void testDelete() throws Exception {

String nif="111111111A";

try {

alumnoDAO.delAlumnos(nif);

}

catch (Exception ex) {

fail("Excepcion no esperada"+ex.getMessage(1));

}

IDatabaseConnection connection = databaseTester.getConnection();

**IDataSet databaseDataSet= connection.createDataSet()**;

**ITable actualTable = databaseDataSet.getTable("alumnos");**

IDataSet expectedDataSet=new FlatXMLDataFileLoader().load(**“/tabla1.xml”**);

**ITable expectedTable = expectedDataSet.getTable("alumnos")**;

Assertion.assertEquals(expectedTable, actualTable);

}

}

**Explica cómo modificarías la configuración del pom del apartado a) para que se detenga la construcción del proyecto si algún test de integración falla (no tienes que volver a copiar el pom).**

**<goals>**

**<goal>integration-test</goal>**

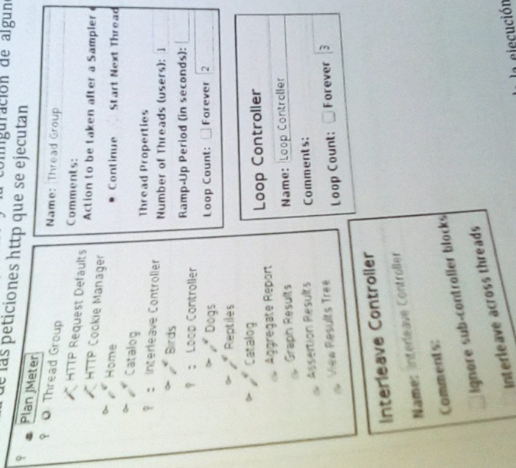
**<goal>verify</goal>**

**</goals>**

**Las fases pre-integration-test . . . verify, por defecto NO tienen asociada ninguna goal cuando el empaquetado del proyecto es jar, por lo que tendremos que incluir y configurar los plugins necesarios en el pom del proyecto.**

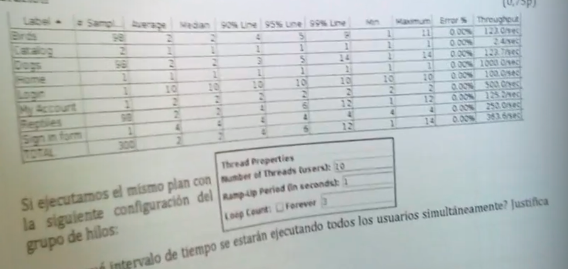
**3.- Contesta las siguientes cuestiones;**

**a) Dado el siguiente plan JMeter y la configuración de algunos de sus elementos indica la secuencia ordenada de las peticiones http que se ejecutan.**

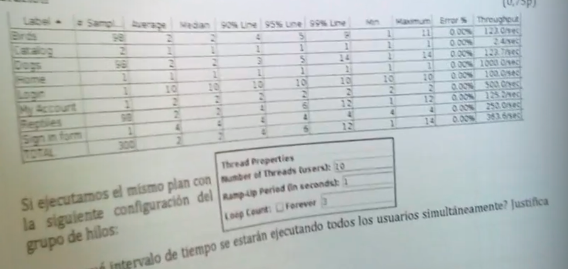


|  |
| --- |
| Escribe aquí la secuencia ordenada |
| 1. **Home** |
| 1. **Catalog** |
| 1. **Birds** |
| 1. **Catalog** |
| 1. **Home** |
| 1. **Catalog** |
| 1. **Dogs** |
| 1. **Dogs** |
| 1. **Dogs** |
| 1. **Catalog** |

**b) Dado el siguiente informe obtenido a partir de la ejecución de un plan con un solo hilo y una única iteración**

****

**Si ejecutamos el mismo plan con la siguiente configuración de hilos:**



**10**

**1**

**3**

**Durante qué intervalo de tiempo se estarán ejecutando todos los usuarios simultaneamente? Justifica la respuesta.**

**Suma del average = 60ms → 0,06s**

**0,06s → Tarda un usuario en abandonar la web**

**El usu0 se**

**va aquí (0,06s)**

**↓**

**usu0 usu1 usu2 usu3 ... usu9**

**|\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|**

**0s 0,1s 0,2s 0,3s ... 1s fin**

**Por lo que, en ningún momento llegan a estar los diez usuarios juntos.**

**c) Ejemplifica con ejemplo correctamente que son las pruebas de carga y las pruebas de stress.**

**4.- Dado el siguiente informe de cobertura de código generado donde el proyecto matriculacion-dao.**

**a) ¿Qué significa el valor N/A en cobertura de líneas y/o condiciones?**

**No aplicable (revisar)**

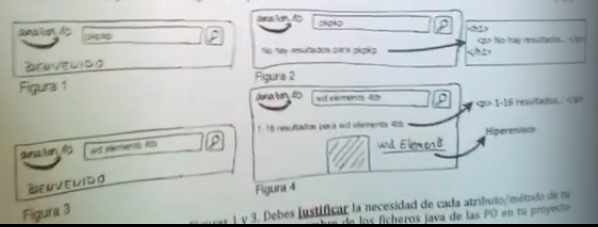
**b) ¿Porque la complejidad ciclomática de algunas clases se representa con un valor decimal?**

**Porque llegamos a mil**

**c) El proyecto matriculacion-dao también ejecuta las clases de matriculacion-comun ¿porque no aparece en el informe?**

**Porque no hay tests asociados a ese proyecto y porque el pom de matriculación-comun no contiene el plugin de jacoco**

**5-. Estamos interesados en implementar dos casos de prueba (usando Maven) sobre la tienda de Amazon.**

****

**En el primero se trata de acceder a la tienda y realizar una búsqueda de un producto que sabemos que no existe en la tienda (ver Figura 1).**

**En cuyo caso obtendremos la respuesta de la Figura 2. Queremos implementar un segundo test en el que busquemos un producto que sí existe en la tienda (ver figura 3) y obtendremos la respuesta de la figura 4. Se pide:**

**a) Implementa la/s PO de las Figuras 1 y 3 Debes justificar la necesidad d ecada atributo/método de tu código. Indica y justifica la ubicación exacta y nombre de los ficheros java de las PO en tu proyecto maven**

public class Figura1**()** **{**

**// Obtenemos todos los elementos de la página**

WebDriver driver**;**

WebElement input**;**

WebElement button**;**

WebElement text**;**

**// Constructor para inicializar los elementos**

public Figura1 **(**WebDriver driver**)** **{**

**this.**driver**=**driver**;**

**this.**driver**.**get**(**"http://amazon.com"**);**

**this.**text**=**driver**.**findElement**(**By**.**cssSelector**(**"h1"**));**

**}**

**// Sacamos el titulo para compararlo y saber si es la página correcta**

public String getTittle **(**Stirng searchContent**)** **{**

**return** text**.**getText**();**

**}**

**// Introducimos el texto que queremos en el input y pulsamos el botón**

public searchInput **(**Stirng searchContent**)** **{**

**this.**input**=**driver**.**findElement**(**By**.**cssSelector**(**"input[type='text']"**));**

**this.**button**=**driver**.**findElement**(**By**.**cssSelector**(**"input[type='submit']"**));**

input**.**sendKeys**(**searchContent**);**

button**.**click**();**

**}**

**}**

public class Figura2 **()** **{**

WebDriver driver**;**

WebElement text**;**

public Figura2**(**WebDriver driver**)** **{**

**this.**driver**=**driver**;**

**this.**text**=**driver**.**findElement**(**By**.**cssSelector**(**"h1>p"**));**

**}**

public String getTittle **(**Stirng searchContent**)** **{**

**return** text**.**getText**();**

**}**

**}**

public class Figura4**()** **{**

WebDriver driver**;**

WebElement text**;**

WebElement link**;**

public Figura4 **(**WebDriver driver**)** **{**

**this.**driver**=**driver**;**

**this.**text**=**driver**.**findElement**(**By**.**tagName**(**"p"**));**

**}**

public String getTittle **()** **{**

**return** text**.**getText**();**

**}**

public String getLink **(**String name**)** **{**

link**=**webElement**.**findElement**(**By**.**linkText**(**name**));**

**}**

**}**

**b) Implementa el primer test (Figuras 1 y 2 ) Nota: Debes mostrar el código completo de la clase que contiene el test (incluyendo el código de los métodos adicionales al anotado con @Test en el caso de que los utilices. Ten en cuenta que debes minimizar la duplicidad de código).**

**Indica y justifica la ubicación exacta del fichero java en tu proyecto maven. Indica y justifica que comando maven debemos usar para ejecutarlo teniendo en cuenta que tipo de pruebas estamos haciendo.**

Src/test/java/examenTest.java

mvn test

webdriver driver**;**

Figura1 f1**;**

Figura2 f2**;**

Figura4 f4**;**

@BeforeEach

public setUp **(){**

driver **=** **new** FirefoxDriver**();**

f1**=** **new** Figura1**(**driver**);**

**}**

@Test

public notFoundProduct**()** **{**

String busqueda="pkpkp";

Assertions**.**assertTrue**(**f1**.**getTittle**.**ToLower**().**contains**(**"BIENVENIDO"**));**

f1**.**searchInput**(**busqueda**);**

f2**=** **new** Figura2**(**driver**);**

Assertions**.**assertTrue**(**snf**.**getTittle**.**ToLower**().**contains**(**"no hay resultados para "**+**busqueda**));**

**}**

@Test

public foundProduct**()** **{**

String busqueda="wd electronics 4tb";

Assertions**.**assertTrue**(**saa**.**getTittle**.**ToLower**().**contains**(**"BIENVENIDO"**));**

String search**=**saa**.**searchInput**(**busqueda**);**

f4 **=** **new** Figura4**(**driver**);**

Assertions**.**assertDoesNotThrow**(**

**()->** f4**.**getLink**(**busqueda**);**

**);**

//o esta

Assertions**.**assertTrue**(**f4**.**getTittle**.**ToLower**().**contains**(**"1 -"**));**

**}**

@AfterEach

public TearDown**(){**

driver.close**();**

**}**

**6.- Estamos interesados en implementar el siguiente caso de pruebas usando Selenium IDE sobre la aplicación Amazon se trata de realizar la búsqueda de un producto y si se encuentra el producto acceder al mismo (si hay varios se elegirá el primero de la lista), y añadirlo a la cesta de compra). Si no se encuentra el producto se debe mostrar el mensaje correspondiente (ver figura 2 del ejercicio anterior).**

**Se pide:**

**a) Explica por qué no puedes generar de forma totalmente automática el test usando botón "recording" y muestra el código de los comandos que necesariamente debes introducir de forma manual para "\_" para representar los comandos que se pueden generar de forma automática antes y después de cada comando manual Justifica claramente dichos comandos explicando exactamente qué hace cada uno de ellos.**

|  |  |  |
| --- | --- | --- |
| Test | | |
| Command | Target | Value |
|  |  |  |
|  |  |  |

**b) ¿Qué tipos de test, de los vistos en esta segunda parte de la asignatura NO puedes realizar con Selenium IDE? Justifica tu respuesta**

**Integración**

**7.- Contesta las siguientes cuestiones.**

**a) Indica si es cierta o falsa la siguiente afirmación "En un plan de pruebas la ejecución de los cuatro tipos de pruebas (unitarias, integración, sistema, y aceptación) se realiza en un orden temporal diferente dependiendo de si se trata de un modelo secuencial, iterativo o ágil".**

**Justifica claramente tu respuesta.**

**Verdadera**

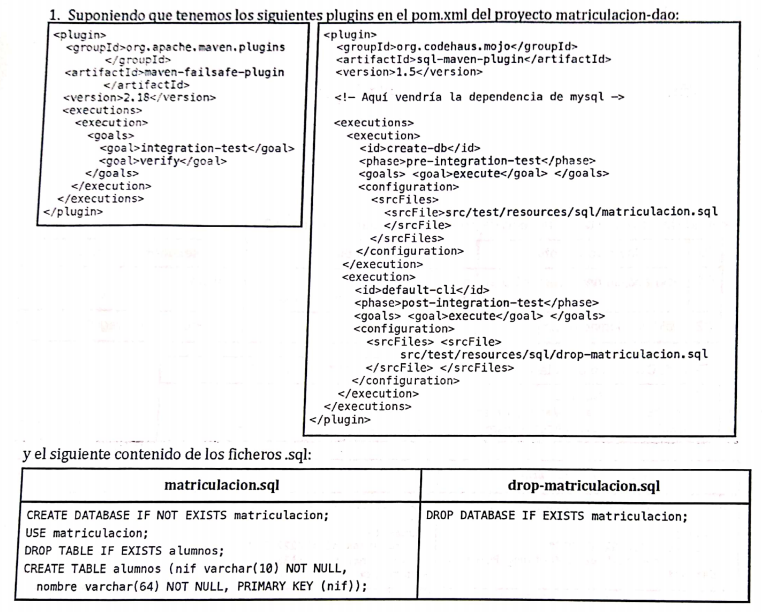
**b) Un plan de pruebas ¿debe contemplar siempre todo los tipos de pruebas vistos en clase? o depende de si estamos una planificación predictiva/adaptativa?**

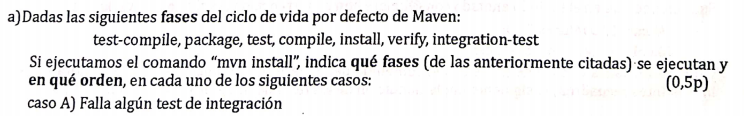
**Justifica claramente tu respuesta.**

**¿Verdadero?**

**RESPUESTAS EXAMEN SEGUNDO PARCIAL 12 DE JUNIO 2018**

**1)**

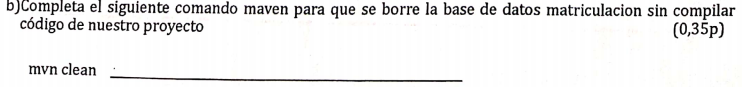
****

****

compile, test-compile, test, package, integration-test, verify(falla en verify)

****

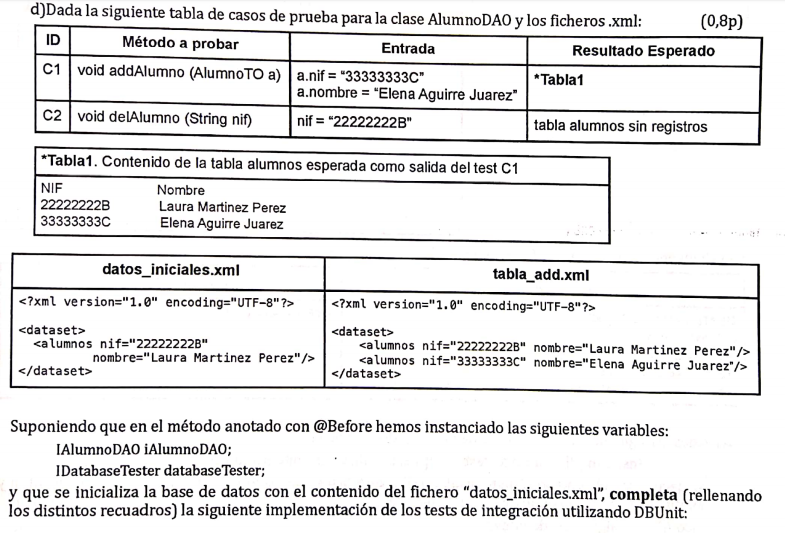
compile, test-compile, test (falla en test)

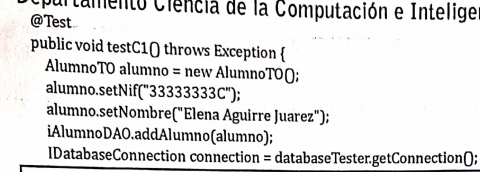


mvn clean sql:execute@default-cli



Sí, se borra la BD aunque haya un error en la fase integration-test la ejecución del proyecto continúa hasta la fase verify.



****

**IDataSet dataSet = new FlatXmlDataFileLoader().load(“datos\_iniciales.xml”);**

**databaseTester.setDataSet(dataSet);**

**databaseTester.onSetup();**

**Assertions.assertDoesNotThrow(()->alumnoDAO.addAlumno(alumno));**

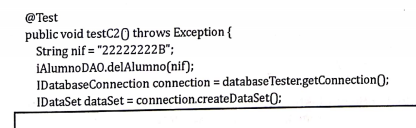
**IDataSet databaseDataSet = connection.createDataSet();**

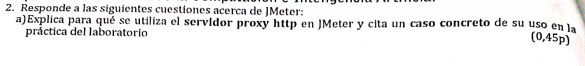
**ITable actualTable = databaseDataSet.getTable(“alumno”);**

**IDataSet expectedDataSet = new FlatXmlDataFileLoader().load(“tabla\_add.xml”);**

**ITable expectedTable = expectedDataSet.getTable(“alumno”);**

**Assertion.assertEquals(expectedTable, actualTable);**

****

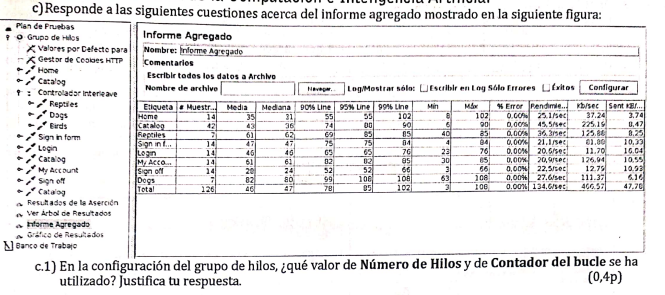
**Lo utilizamos para poder conocer los datos que enviamos, por ejemplo cuando enviamos por POST en un login.**

****

**Consejo 1: No tener abierto/ejecutando otros procesos.**

**Consejo 2: Hacer las pruebas en distintas máquinas.**

**Consejo 3: Dejar una prueba JMeter ejecutándose durante largos periodos de tiempo.**

****

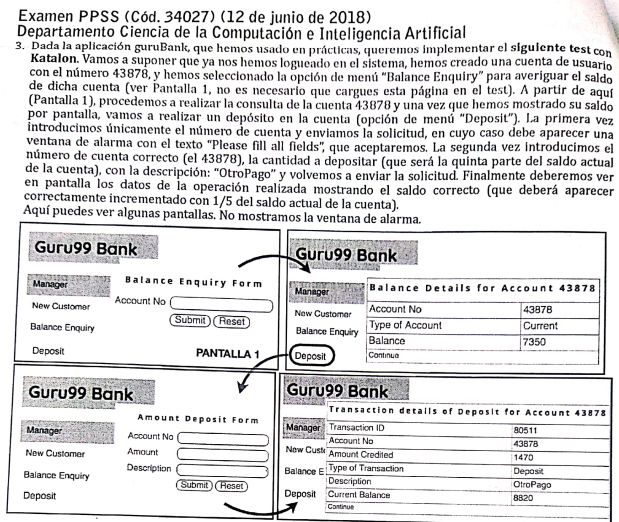
**Hilos: 7, Bucle: 2**

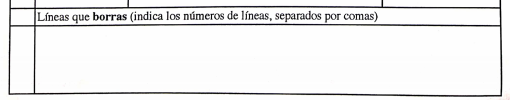
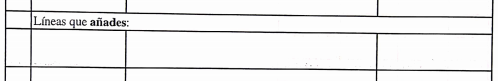
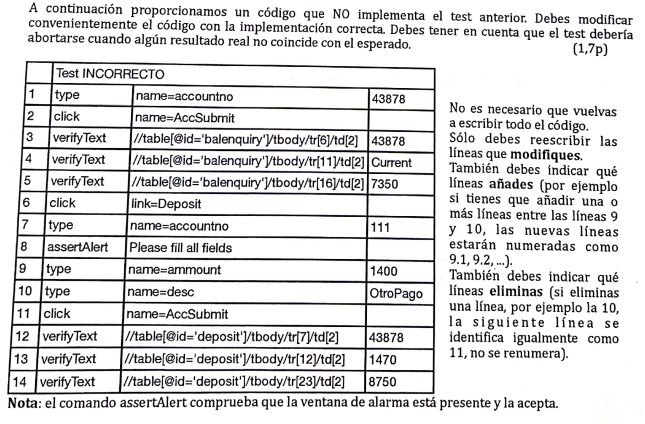
****

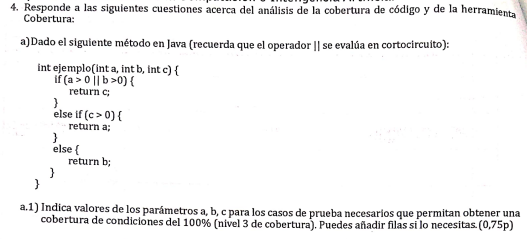
**catalog 45,5 segundos**

****

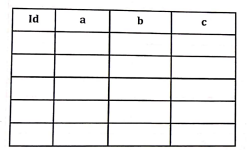
**sign off 20 / 1000 = xxxx**

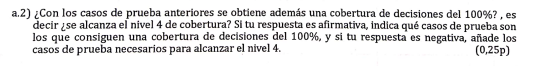
****

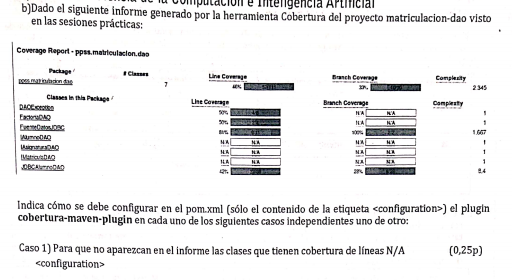
****

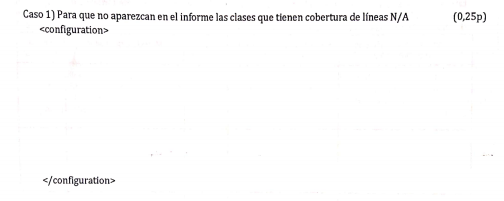
****

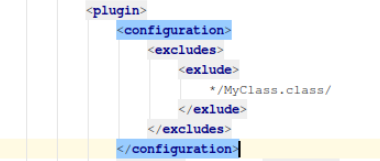
****

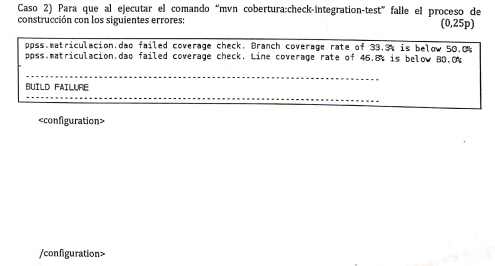
****

****

****

****



****

***<rules>***

***<rule>***

***<element>BUNDLE</element>***

***<excludes>***

***<exclude>IAlumnoDAO</exclude>***

***<exclude>IAsignaturaDAO</exclude>***

***<exclude>IMatriculaDAO</exclude>***

***</excludes>***

***<limits>***

***</limits>***

***</rule>***

***</rules>***

**<rules>**

**<rule>**

**<element>BUNDLE</element>**

**<limits>**

**<limit>**

**<counter>BRANCH</counter>**

**<value>COVEREDRATIO</value>**

**<minimum>50%</minimum>**

**</limit>**

**</limits>**

**</rule>**

**<rule>**

**<element>BUNDLE</element>**

**<limits>**

**<limit>**

**<counter>LINE</counter>**

**<value>COVEREDRATIO</value>**

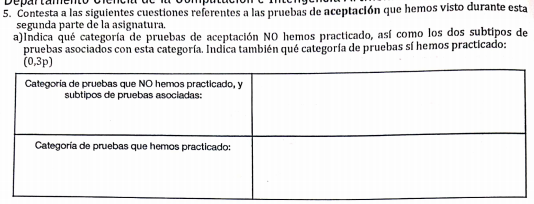
**<minimum>80%</minimum>**

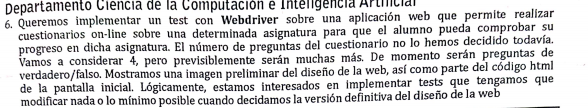
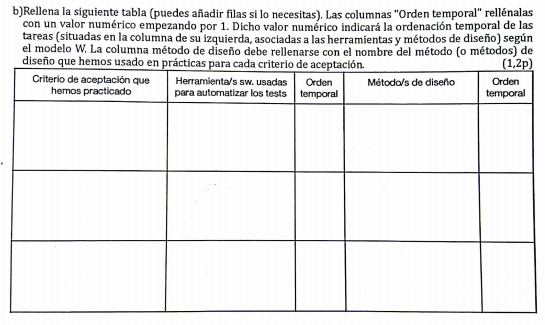
**</limit>**

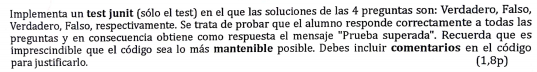
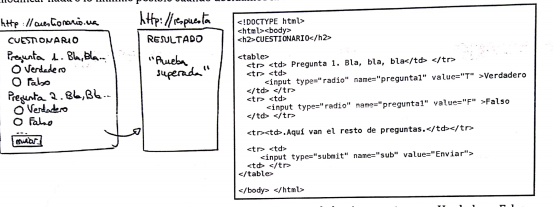
**</limits>**

**</rule>**

**</rules>**

****

****

****

**@Test**

**public void t1() {**

**webdriver driver = new FirefoxDriver();**

**driver.get("**[**http://cuestionario.me**](http://cuestionario.me)**");**

**WebElement tittleQuestion = driver.findElement(By.TagName("h2"));**

**Assertions.assertTrue(tittleQuestion.getText().toLower().contains("cuestionario") );**

**boolean respuestas[] = {true,false,true,false};**

**List<Webelement> input = driver.findElements(By.TagName("input"));**

**Webelement button = driver.findElement(By.cssselector("input[type='submit']"));**

**for (var i=0; respuestas.size(); i++) {**

**int sum=0;**

**if (respuesta[i]==false) {**

**sum=1;**

**}**

**input.get(i\*2+sum).click();**

**}**

**button.click();**

**Webelement tittle = driver.findElement(By.tagName("h1"));**

**Assertions.assertTrue(tittle.getText().toLower().contains("prueba superada") );**

**}**

**--------------------------------------------------------------------------------**

**@Test**

**public void test(){**

**WebDriver driver = new FirefoxDriver(); //Creo el navegador**

**driver.get("http://cuestionario.ua"); //le asigno la pagina web**

**//obtengo el titulo**

**WebElement tituloPag1 = driver.findElement(By.cssSelector("h2"));**

**//compruebo q estoy donde quiero estar**

**Assertions.assertTrue(tituloPag1.getText().contains("CUESTIONARIO"));**

**List<WebElement> listaT = driver.findElements(By.cssSelector("input[value='T']"));**

**List<WebElement> listaF = driver.findElements(By.cssSelector("input[value='F']"));**

**//Inicializo el array y le doy valores**

**Boolean[] boolArray = new Boolean[4];**

**boolArray[0] = true;**

**boolArray[1] = false;**

**boolArray[2] = true;**

**boolArray[3] = false;**

**for(int i = 0;i < boolArray.length;i++){ //Esto marca Verdadero y Falso**

**if(boolArray[i] == false)**

**listaF.get(i).click();**

**else**

**listaT.get(i).click();**

**}**

**WebElement boton = driver.findElement(By.name("sub"));**

**boton.click(); //pinchamos en el boton de "enviar"**

**//ASUMIMOS QUE EL RESULTADO ESTA EN UN H3 Y ES EL UNICO**

**WebElement resultado = driver.findElement(By.cssSelector("h3"));**

**//obtengo el resultado**

**Assertions.assertTrue(resultado.getText().contains("Prueba superada"));**

**}**

**PREGUNTAS EXAMEN 2017**

**Ejercicio 1:**

**a. Atendiendo a la frase "Puede existir una cobertura de nivel 3, sin que exista cobertura de nivel 1", demuestra con un ejemplo concreto de código esta afirmación.**

**El nivel 3 es cobertura de condiciones (100%), y nivel 1 es cobertura de líneas (100%).**

**Un 100% de cobertura de condiciones NO garantiza un 100% de cobertura de líneas.**

**Sí, por ejemplo:**

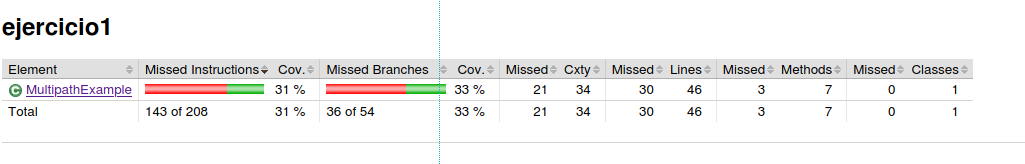
**if (a&&b) {**

**a=true;**

**}**

**Conseguimos el nivel 3 con a=false, b=true, y luego a=true, b=false. Pero no entramos a la línea de dentro del if.**

**b. El informe de Cobertura nos da tres medidas. Cítalas y explicalas.**

****

**Nos proporciona información sobre las líneas no testeadas, las ramas no testeadas y la complejidad ciclomática.**

**Ejercicio 2:**

**a. Un pantallazo de Grupo de Hilos de JMeter, explicar con un ejemplo concreto cada uno de los campos del apartado propiedades.**

**b. Un pantallazo de un proyecto JMeter, y explicar los Elementos (Grabado, Interleave, Listeners, Samplers y Assertions)**

**Ejercicio 3:**

**Hacer con Webdriver dos tests. Uno con Page Object Pattern y otro sin él. Explicar la diferencias y las ventajas de usar Page Object.**

**Las diferencias de usar o no Page Object son:**

1. **Facilita la mantenibilidad.**
2. **Reducir la duplicación de código de los tests.**

**La diferencia es a la hora de seleccionar los elementos de la página.**

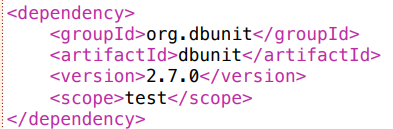
**Ejercicio 4:**

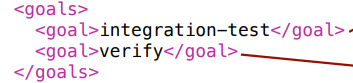
**Una tabla con comandos Selenium y corregirlos.**

**Ejercicio 5:**

**a. Te marcaban unas lineas de código DBunit y tenías que explicar lo que significaba cada método dentro del contexto de prácticas. Además explicar cada argumento si lo tenía.**

**b. Citar el artefactId usado para prácticas de DBunit y sus goals y porqué los usamos aquí.**

****

****

**Ejercicio 6:**

**tipo test justificado del tema 13. 3 preguntas de modelos iterativos y secuencial, bastante sencillo.**

**Ejercicio 7. (Edito)**

**a. Si estaba relacionado las features y la historia.**

**b. La relación entre los escenarios y los steps.**

**PREGUNTA 1**

**a) ¿Porqué hay que incluir lo de la imagen en el pom de los proyectos con empaquetado “jar”? (0’2)**

**b) Ordenar las siguientes fases en el orden en el que se ejecutarían al hacer el “mvn install” en estos casos: (0’7)**

**Fases: compile, test-compile, test, integration-test, install, verify, package (y no se si había alguno más)**

**● Si no hay ningún fallo en la ejecución**

**● Si falla algún test de integración**

**● Si falla algún test unitario**

**PREGUNTA 2**

**a) ¿Para qué sirve el proxy? (0’15)**

**b) ¿Qué pasos hay que seguir para usar el proxy?**

**c) ¿Cómo hemos usado el servidor proxy en las prácticas?**

**PREGUNTA 3**

**Pon las peticiones que se realizarán, en el orden correcto.**

**PREGUNTA 4**

**Un dibujo esquemático con dibujos de páginas web, habían 3 páginas (el usuario ya está registrado, no hay que hacer login ni nada):**

**● Una página de inicio con 3 linkText, donde uno es “Leer Correo”**

**● Cuando le das a “Leer Correo”, te lleva a la página de correo del usuario. En este caso tenemos 2 opciones, A y B, con diferentes diseños de la página pero con los mismos elementos, 4 linkText: spam, correo y no me acuerdo (nos importa spam), y al lado de cada linkText un número que representa la cantidad de ese tipo de mensajes. La página también tiene un botón ”Back”.**

**● Al darle a “spam” nos lleva a otra página donde nos salen los mensajes de spam, además de una opción de seleccionar todo, dónde todo tiene un ¿checkbox o select? para seleccionarlo. La página también tiene botón de “Borrar” y “Back”. Si no hay spam, solo sale un texto diciendo que no tienes mensajes.**

**a) El enunciado pedía hacer el driver para probar el borrado de todos los mensajes de spam, pero sin saber aún si se usará el diseño A o B de la página mencionada antes. Además también dice que para implementar el driver uses una de las dos formas vistas en clase y justifiques el porqué.**

**b) Se ha decidido hacer la opción A, decir si se podría usar Selenium IDE para hacer el driver en lugar de WebDriver y justificarlo.**

**c) No me acuerdo**

**PREGUNTA 5**

**Pregunta de DbUnit, donde te pide implementar el código que falta en una clase de test.**

**Los test son uno de borrar un alumno y otro de obtener un alumno de una tabla alumnos de una BD, y te daba los xml con la situación inicial y el resultado esperado del borrar.**

**PREGUNTA 6**

**Pregunta sobre el modelo W de planificación de pruebas de un proyecto basado en desarrollo secuencial, poniendo las tareas, las herramientas usadas, etc. Pero de lo que hemos hecho en la segunda parte del cuatrimestre.**

**PREGUNTAS ALBERTO**

**1)En el código de pruebas, explica qué representan las variables *\_customerFactory* y *databaseTester* indicando para qué y dónde se utilizan.**

**\_customerFactory:** Esta variable almacena la clase CustomerFactory creado de esa forma para permitir pruebas unitarias

**databaseTester:** Esta variable almacena la clase JdbcDatabaseTester que tiene una conexión con la BD.

**2) En el código de pruebas, explica la diferencia entre un dataset y una table. Identifica dónde y para qué se utilizan en el código proporcionado.**

El dataset se obtiene en ambos test y se usa para obtener todos los datos de la base de datos o xml. Una vez obtenidos, en uno de los tests es necesario comprobar que se ha introducido correctamente, por lo que obtiene la tabla específica y compara que ha quedado como el resultado esperado.

Nota: las librerías descargadas por maven en nuestro repositorio local son necesarios para construir el proyecto, pero NO son artefactos generados por Maven durante la construcción del proyecto

**3) ¿Para qué sirven las librerías para usar junit5 (junit-jupiter-engine), y los plugins surefire y failsafe?**

El plugin **junit5** es para poder usar JUNIT y DBUNIT, ya que este segundo es una extensión del primero.

El plugin **surefire** es para ejecutar los test unitarios.

El plugin ***failsafe*** es para ejecutar los tests de integración y verify.

**4¿Que pasa cuando se ejecuta un proyecto padre con módulos?**

Cuando se ejecuta el proyecto padre que contiene distintos módulos, ejecuta primero el principal (matriculacion) y después va ejecutando de forma ordenada (matriculacion-comun y los otros) según sean más independientes. Como matriculacion-comun no depende de ninguno es el primero en ejecutarse, y luego van los siguientes según tengan más o menos dependencias. Cuando ejecuta uno de estos, ejecuta todas las fases hasta llegar a la indicada. En este ejemplo he ejecutado *“mvn verify”,* por lo que ejecutará desde la fase compile hasta verify, luego hará lo mismo con matriculación-común, luego con matriculación-dao y así con todas.

**5 ¿Qué pasa si no ejecutamos "cobertura:clean"?**

Que se siguen sumando a las pruebas ya hechas anteriormente. Por lo que la información se acumula y podría no ser correcto el informe que veamos.

**6 ¿Podemos utilizar "clean" en lugar de "cobertura:clean"? ¿Cuál es la diferencia?**

No estoy seguro, pero diría que sí. **Cobertura:clean** reinicia el informe de cobertura. El comando **clean** sirve para borrar todos los archivos generados en la construcción del proyecto y debido a que elimina la carpeta target también eliminaría el informe.

**7 ¿Cuál es la complejidad ciclomática (CC) del método multiPath()? Implementa un número mínimo de casos de prueba para conseguir una cobertura del 100% de líneas y de condiciones. ¿Cuál es ese número? Explica por qué es diferente del valor de CC.**

## 

Cobertura de líneas solo busca recorrer todas las líneas, y la complejidad ciclomática va más orientado a la complejidad para recorrer todos los caminos. Con a=6 y b=6 cumpliriamos la cobertura de líneas.

La CC es 3, por lo que necesitaríamos dos pruebas para recorrer todos los caminos. Por ejemplo: a=4/b=4 y a=6/b=6.

**7 ¿Cuáles son las dos formas de obtener un informe de cobertura vistas en clase de teoría?. Indica las diferencias entre ambas opciones observando las goals ejecutadas que se muestran en consola de salida de Maven.**

* mvn cobertura:cobertura → Se ejecuta en su propio ciclo de vida (instrumenta el código, ejecuta los tests y genera un informe).
* mvn site → Genera dos informes de cobertura en formato html: cobertura (de los tests unitarios) y cobertura-integration-test (de los tests de integración).

Dependiendo de la configuración del pom, se puede usar una alternativa u otra. Pero al hacerlo con site no se debe especificar de qué tests quieres sacar el informe, ya que te genera ambos.

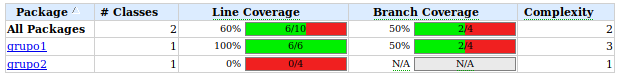
**8 ¿Qué debes hacer para que, ejecutando la alternativa 1 (**mvn cobertura:cobertura)  **y a continuación la 2 (**mvn site ) **o al revés, obtengas exactamente el mismo informe?**

Se debe hacer un clean (cobertura:clean) antes de ejecutar la siguiente alternativa, porque sino en el informe la cobertura se va acumulando a la anterior. De esta forma, el contador de las sentencias ejecutadas se resetearía.

**9 ¿Cómo ejecutar el comando mvn site sin tener que recompilar todo?**

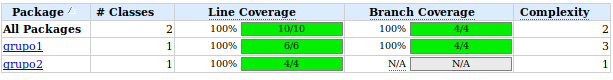
.-------

**10 ¿Que pasa si no tenemos tests de para del código?**

******

En la parte del grupo2 es lo que obtendremos si no tenemos código de test.

Una vez resueltos obtenemos lo siguiente:

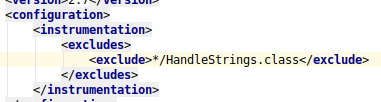


N/A, en este caso, significa que no tenemos condiciones en el código.

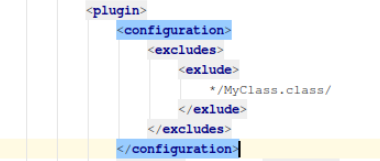
**11 Explica qué diferencia hay entre el valor N/A**  **y cero**.

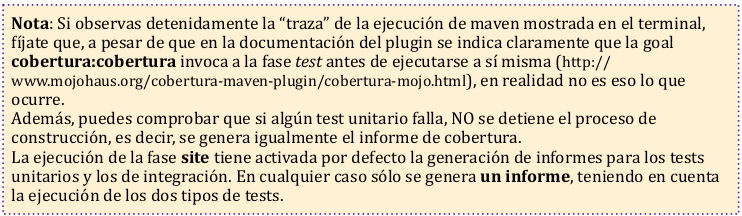
No puede ser 0 debido a que sea verdadero o falso la condición, se prueban una de sus vertientes.

Para ignorar una clase es necesario: (ELIGE UNA, LA SEGUNDA ES MEJOR)



---------------------------------------------------------------------------------------------------------





## Notas:

* <haltOnFailure>true</haltOnFailure>. Si no se pone está por defecto en true. Si está a true y ponemos el resto de condiciones en la etiqueta <check> en el pom, se parará la construcción del proyecto si no se cumples estas condiciones y si está a false, no se parará.
* Probablemente pregunte lo de las etiquetas check para decir la cobertura de líneas y condiciones.

**12 ¿Qué criterios debe satisfacer el sistema para ser aceptado por el cliente?**

* + Un principio básico para diseñar los criterios de aceptación es asegurar que la calidad del sistema es aceptable
  + Los criterios de aceptación deben ser medibles y cuantificables

**13 ¿Cuando usamos DBUnit para los test de integración, nuestro test accede a la base de datos REAL?** Sí, accede directamente a la real.

**Preguntas de Prácticas**

**Práctica7:**

**1.B) En el código de pruebas, explica qué representan las variables clienteDAO y databaseTester indicando para qué y dónde se utilizan.**

ClienteDAO: Es la clase que utilizamos para interactuar con la BD, para realizar los inserts, los deletes, los upgrades y el read.

databaseTester: Es la interfaz que nos permite el acceso a la BD.

**1.C) En el código de pruebas, explica la diferencia entre un dataset y una table. IdentiFica dónde y para qué se utilizan**

Dataset: Es un conjunto de tablas con sus respectivos datos.

Tabla: Es donde se almacenan los datos.

La diferencia es que un Dataset contiene una serie de tablas, dicho dataset es utilizado para poblar la BD con datos.

Utilizamos los datasets para poblar la base de datos, a continuación ejecutamos el SUT y almacenamos los resultados de la tabla modificada por dicho SUT en una tabla auxiliar.  
A continuación insertamos el dataset con los resultados esperados, obtenemos la tabla esperada y comparamos dicha tabla con la tabla con los datos modificados por el SUT.

--**1.D)**

· Las goals que se han ejecutado para construir el proyecto y los artefactos generados en target han sido:

- process-resources: maven-resources-plugin:2.6:resources (default-resources)

· Como en src/main/java/resources no hay nada, entonces no copia nada en target

- compile: maven-compiler-plugin:3.1:compile (default-compile)

· Compila los 3 ficheros y los .class se guardan en target/classes/ppss/\*.class

· Se crea el directorio generated-sources/annotations vacío ya que no hay anotaciones

· Se crea el directorio maven-status/maven-compiler-plugin/compile/default-compile/\*.lst

- process-test-resources: maven-resources-plugin:2.6:testResources (default-testResources)

· Copia los resources de los test en target/test-classes/\*

· Se crea el directorio generated-test-sources/test-annotations vacío ya que no hay anotaciones

- test-compile: maven-compiler-plugin:3.1:testCompile (default-testCompile)

· Compila el fichero y el .class se guarda en target/test-classes/ppss/\*\_IT.class

· Se crea el directorio maven-status/maven-compiler-plugin/testCompile/default-testCompile/\*.lst

- test: maven-surefire-plugin:2.22.2:test (default-test)

- package: maven-jar-plugin:2.4:jar (default-jar)

· Crear en target/maven-archiver/pom.properties

· Crea el .jar en target/dbunitexample-1.0-SNAPSHOT.jar

- pre-integration-test: sql-maven-plugin:1.5:execute (create-customer-table)

- integration-test: maven-failsafe-plugin:2.22.2:integration-test (default)

· Se crea el informe en target/failsafe-reports/\*.xml y \*.txt

- verify: maven-failsafe-plugin:2.22.2:integration-test (default)

**2.A) Indica para qué sirven exactamente para que sirve surefire, failsafe, sql y la librería dbunit:**

Surefire: Es el plugin utilizado para llevar a cabo las pruebas unitarias. Actúa en la fase test del ciclo de vida build, ejecutando los unit tests.

Failsafe: Es utilizado para ejecutar las pruebas de integración, se deberá asociar la goal “integration-test” a la fase con el mismo nombre.

Plugin sql: Se utiliza para poder ejecutar sentencias SQL.

Dbunit: Se utiliza la librería dbunit para automatizar las pruebas de integración con una BD. Además de controlar el estado de la BD antes de las pruebas y compararlas con el resultado de después de ejecutar dichas pruebas.

**2.I) Añade también tests unitarios y de integración en los proyectos matriculacion-bo y matriculacionproxy.**

**Luego ejecuta mvn verify desde el proyecto padre. Anota la secuencia en la que se ejecutan los**

**diferentes tipos de tests de todos los módulos. ¿Qué estrategia de integración estás siguiendo?**

Ejercicio 2.I

· Secuencia (mvn verify):

- Building matriculacion 1.0-SNAPSHOT

· maven-failsafe-plugin:2.22.2:integration-test

· maven-failsafe-plugin:2.22.2:verify

- Building matriculacion-comun

· maven-resources-plugin:2.6:resources

· maven-compiler-plugin:3.1:compile

· Compiling 5 source files

· maven-resources-plugin:2.6:testResources

· maven-compiler-plugin:3.1:testCompile

· maven-surefire-plugin:2.22.2:test

· maven-jar-plugin:2.4:jar

· maven-failsafe-plugin:2.22.2:integration-test

· maven-failsafe-plugin:2.22.2:verify

- Building matriculacion-dao 1.0-SNAPSHOT

· maven-resources-plugin:2.6:resources

· maven-compiler-plugin:3.1:compile

· Compiling 9 source files

· maven-resources-plugin:2.6:testResources

· Copying 6 resources

· maven-compiler-plugin:3.1:testCompile

· Compiling 2 source files

· maven-surefire-plugin:2.22.2:test

· Tests run: 3, Failures: 0, Errors: 0, Skipped: 0

· maven-jar-plugin:2.4:jar

· sql-maven-plugin:1.5:execute

· 5 of 5 SQL statements executed successfully

· maven-failsafe-plugin:2.22.2:integration-test

· Tests run: 7, Failures: 0, Errors: 0, Skipped: 0

· maven-failsafe-plugin:2.22.2:verify

- Building matriculacion-proxy 1.0-SNAPSHOT

· maven-resources-plugin:2.6:resources

· maven-compiler-plugin:3.1:compile

· Compiling 2 source files

· maven-resources-plugin:2.6:testResources

· maven-compiler-plugin:3.1:testCompile

· Compiling 2 source files

· maven-surefire-plugin:2.22.2:test

· Tests run: 3, Failures: 0, Errors: 0, Skipped: 0

· maven-jar-plugin:2.4:jar

· maven-failsafe-plugin:2.22.2:integration-test

· Tests run: 3, Failures: 0, Errors: 0, Skipped: 0

· maven-failsafe-plugin:2.22.2:verify

- Building matriculacion-bo 1.0-SNAPSHOT

· maven-resources-plugin:2.6:resources

· maven-compiler-plugin:3.1:compile

· Compiling 4 source files

· maven-resources-plugin:2.6:testResources

· maven-compiler-plugin:3.1:testCompile

· Compiling 2 source files

· maven-surefire-plugin:2.22.2:test

· Tests run: 3, Failures: 0, Errors: 0, Skipped: 0

· maven-jar-plugin:2.4:jar

· maven-failsafe-plugin:2.22.2:integration-test

· Tests run: 3, Failures: 0, Errors: 0, Skipped: 0

· maven-failsafe-plugin:2.22.2:verify

· Estrategia de integración:

1. Matriculacion-comun

2. Matriculacion-dao -> Capa de acceso a datos (Implementación de DAOs)

3. Matriculacion-proxy -> Capa de negocio

4. Matriculacion-bo -> Capa de negocio (cliente de la capa de acceso a datos)

- Bottom-up: primero se integran los componentes con menor nivel de abstracción (acceso a datos) y

después se añaden los componentes funcionales, cada vez con mayor nivel de abstracción